UNIT-VI

Graphs

A graph is an abstract data structure that is used to implement the mathematical concept of graphs.

→ Graph is basically a collection of vertices (also called nodes) and edges that connect these vertices.

→ Graphs are widely used to model any situation where entities or things are related to each other in pairs. for ex., Transportation networks in which nodes are airports, intersections, ports etc. The edges can be airline flights, one-way roads, shipping routes etc.

Definition: A graph G is defined as an ordered set (V, E) where V(G) represents the set of vertices and E(G) represents the edges that connect these vertices.

ex:

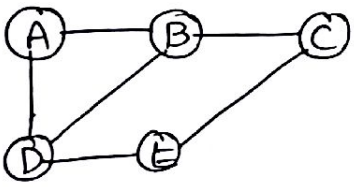


fig: Undirected graph.

$V(G) = \{A, B, C, D, E\}$

$E(G) = \{(A,B), (B,C), (A,D), (B,D), (D,E), (C,E)\}$

→ Graph
- directed graph
- undirected graph.

Undirected graph :- In an undirected graph, edges do not have any direction associated with them. That is, if an edge is drawn b/n nodes A & B, then the nodes can be traversed from A to B as well as from B to A.

ex:

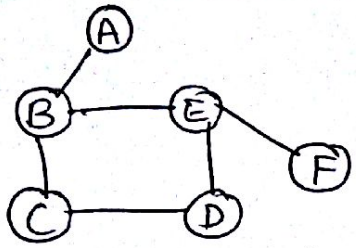


fig: Undirected graph

Directed graph: A directed graph or digraph is a graph in which each line has a direction (arrow head) to its successor. If there is an edge from A to B, then there is a path from A to B but not from B to A.

ex:

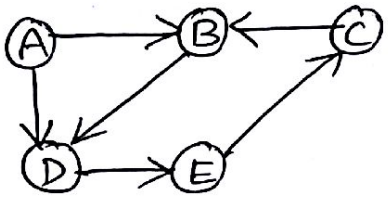


fig: Digraph.

Graph Terminology :-

① Adjacent nodes or neighbours: For every edge e=(u,v) that connects nodes u and v, the nodes u, v are the end-points and are said to be the adjacent nodes or neighbours.

ex: In the above digraph, B to adjacent to A
D to adjacent to E

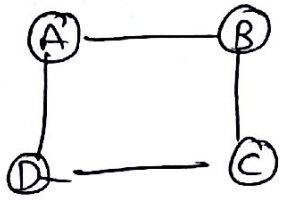
② Degree of a node: Degree of a node u, deg(u), is the total no. of edges containing the node u. If deg(u)=0, it means that u does not belong to any edge and such a node is known as an isolated node.

ex: Degree of E in the above undirected graph is 3

(3) Regular graph: It is a graph where each vertex has the same number of neighbours. That is, every node has the same degree.

A regular graph with vertices of degree k is called a k-regular graph or a regular graph of degree k.

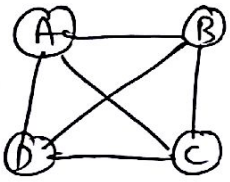
ex:



degree of A = 2
degree of B = 2
degree of C = 2
degree of D = 2

(4) Path: A path is a sequence of vertices in which each vertex is adjacent to the next one.

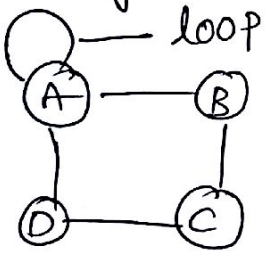
ex:



{A, B, C, D} is one path.
{A, C, B, D} is another path.

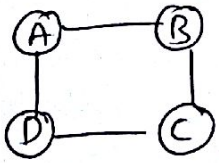
(5) Closed path or loop: A path p is known as a closed path if the edge has the same end points.

ex:



(6) Cycle: A path in which the first and last vertices are same. A simple cycle has no repeated edges or vertices (except the first and last vertices).

ex:



{A, B, C, D, A} is a cycle.

(7) Connected graph: A graph is said to be connected, if there is a path from any vertex to any other vertex.

→ Two vertices v_i, v_j in a graph G are said to be connected only if there is a path in G b/n v_i & v_j.

→ In an undirected graph if v_i & v_j are connected then it automatically holds that v_j and v_i are also connected

→ An undirected graph is said to be a connected graph if every pair of distinct vertices v_i, v_j are connected.

ex:

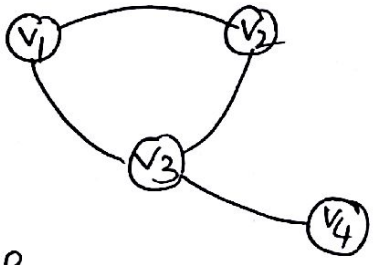


fig: Undirected - connected graph.

→ A directed graph can be strongly connected, weakly connected & disjoint graph.

Strongly connected:- A directed graph is said to be strongly connected if every pair of distinct vertices v_i, v_j are connected (by means of a directed path). Thus if there exists a directed path from v_i to v_j then there also exists a directed path from v_j to v_i.

ex:

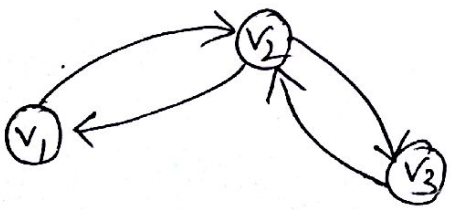


fig: Stongly connected graph

Weakly connected: In a digraph, atleast 2 vertices are not connected.

ex:

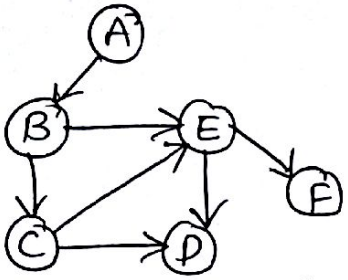
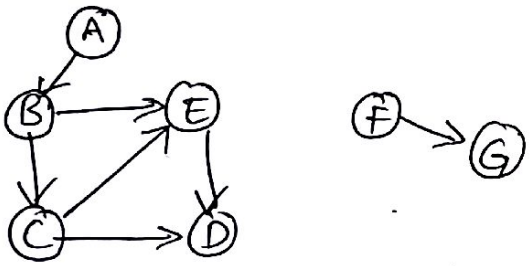


fig: Weakly connected.

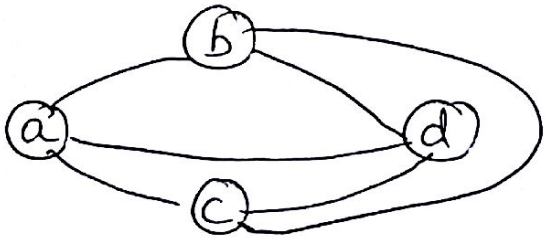
Disjoint graph: A graph is a disjoint graph, if it is not connected.

ex:



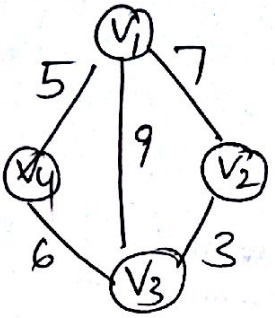
⑧ Complete Graph: A graph G is said to be complete if all its nodes are fully connected. i.e., there is a path from one node to every other node in the graph. A complete graph has $n(n-1)/2$ edges, where n is the no. of nodes in G.

ex:



⑨ Weighted graph or Labelled graph: A graph is said to be weighted graph if all the edges in it are labelled with some weights.

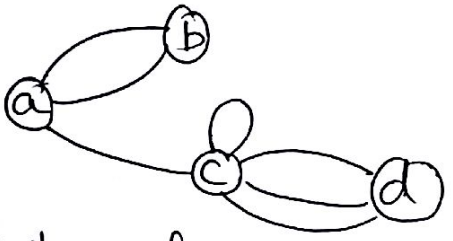
ex:



(10) Multi graph : A graph with multiple edges and/or loops is called as multi graph.

→ Distinct edges which connect the same end-points are called multiple edges.

ex:



Representation of Graphs :- There are several representations are possible to store the graphs in computer memory.

1. Adjacency matrix
2. path matrix
3. circuit matrix
4. cutset matrix
5. Adjacency list
6. Adjacency multi list
7. Incident matrix

1. Adjacency Matrix: The adjacency matrix is used to represent the nodes which are adjacent to one another. The adjacency matrix of a graph 'G' with 'n' vertices is form

'nxn' symmetric binary matrix as,

$$a_{ij} = \begin{cases} 1 & \text{if } i \text{ \& } j \text{ vertices are adjacent to each other} \\ 0 & \text{otherwise} \end{cases}$$

ex:

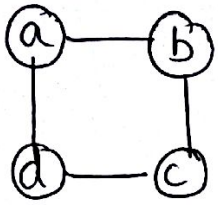


fig: Undirected graph

Adjacency matrix is,

	a	b	c	d
a	0	1	0	1
b	1	0	1	0
c	0	1	0	1
d	1	0	1	0

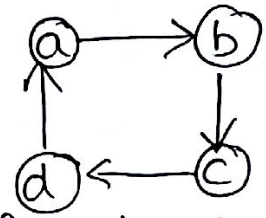


fig: directed graph

Adjacency matrix is,

	a	b	c	d
a	0	1	0	0
b	0	0	1	0
c	0	0	0	1
d	1	0	0	0

2. Incidence Matrix:- Let G be a graph with n vertices and e edges. Define an 'nxe' matrix $M = [m_{ij}]$ whose n rows correspond to n vertices and e columns correspond to e edges as,

$m_{ij} = 1$ if the j^{th} edge e_j is incident on the i^{th} vertex v_i

$= 0$ otherwise

Matrix M is known as the incidence matrix representation of the graph G.

ex:

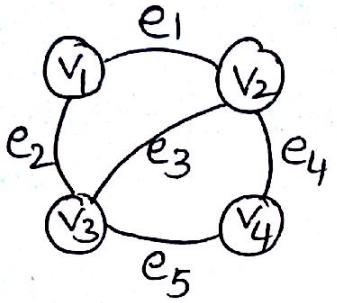


fig: Undirected graph

$$M = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

fig: Incidence matrix

3. Circuit Matrix:- For a graph G let the no. of different circuits be t and the no. of edges be e. Then the circuit matrix, $C = [C_{ij}]_{txe}$

$C_{ij} = 1$ if the i^{th} circuit includes the j^{th} edge

$= 0$ otherwise

ex:

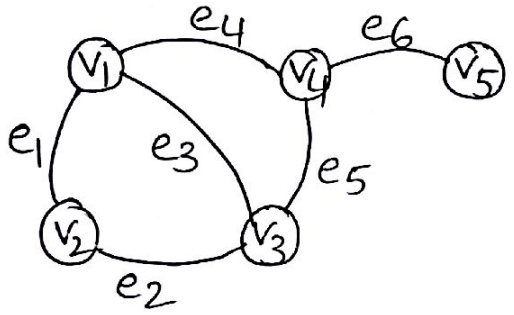


fig: Undirected graph

→ The circuits are expressed in terms of their edges:

(C1) circuit 1 : $\{e_1, e_2, e_3\}$

(C2) circuit 2 : $\{e_3, e_4, e_5\}$

(C3) circuit 3 : $\{e_1, e_2, e_5, e_4\}$

The circuit matrix is,

$$C_{3\times 6} = \begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

4. Cut set Matrix :- For a graph G, a matrix $S = [S_{ij}]$ whose rows correspond to cut sets and columns correspond to edges of the graph is defined to be a cut set matrix if $S_{ij} = 1$, if the i^{th} cut set contains the j^{th} edge

$= 0$, otherwise

ex:

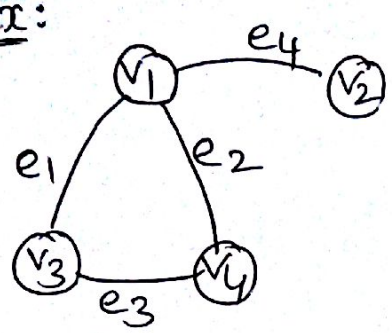
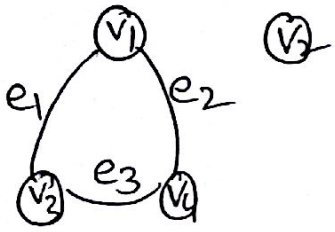


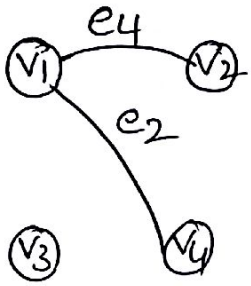
fig: Undirected graph

Cutset: A cutset in a connected graph G is the set of edges whose removal from G leaves G disconnected.

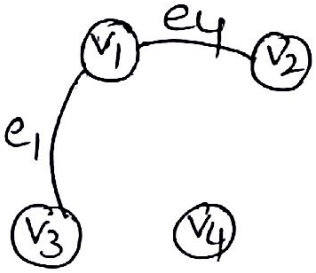
ex: The cutsets of the above graph are,



$S_1 = \{e_4\}$



$S_2 = \{e_1, e_3\}$



$S_3 = \{e_2, e_3\}$

~~$S_4 = \{e_3, e_4\}$~~

$S_4 = \{e_1, e_2\}$

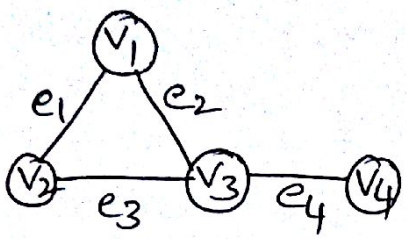
Cutset matrix is,

	e_1	e_2	e_3	e_4
S_1	0	0	0	1
S_2	1	0	1	0
S_3	0	1	1	0
S_4	1	1	0	0

5. Path Matrix: A path matrix is generally defined for a specific pair of vertices in a graph. If (u, v) is a pair of vertices then the path matrix denoted as $P(u,v) = [P_{ij}]$ is given by $P_{ij} = 1$, if j^{th} edge lies in the i^{th} path b/n vertices u & v

$= 0$, otherwise

ex:



The paths b/n vertices V_1 & V_4

P_1 : $\{e_2, e_4\}$

P_2 : $\{e_1, e_3, e_4\}$

fig: Undirected graph

The path matrix representation is,

$$\begin{array}{c} \\ P_1 \\ P_2 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

6. Adjacency List :- The linked representation of graphs is refered to as adjacency list representation and is comparatively efficient with regard to adjacency matrix representation.

Given a graph G with n vertices and e edges, the adjacency list opens n head nodes corresponding to the n vertices of graph G, each of which points to a singly linked list of nodes, which are adjacent to the vertex representing the head node.

ex:

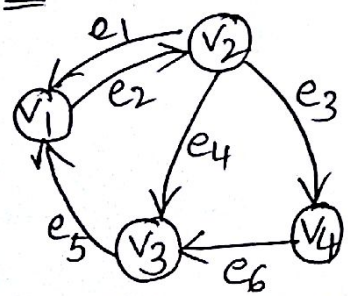


fig: Digraph

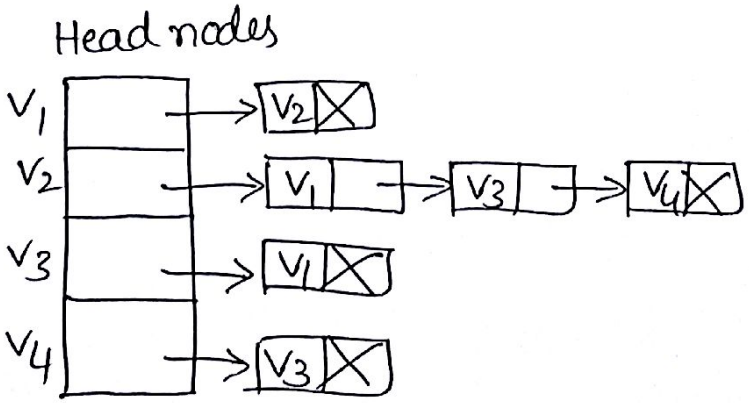
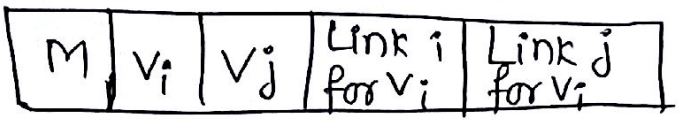


fig: Adjacency list representation

7. Adjacency Multi list: This is the modified version of adjacency lists. Adjacency multi-list is an edge based rather than a vertex based representation of graphs. The structure of multi list consists of 5 parts.



where, M – A single bit field to indicate whether the edge has been examined or not.

V_i – A vertex in the graph i.e., connected to vertex V_j by an edge

V_j – A vertex in the graph i.e., connected to vertex V_i by an edge.

Link i for V_i – A link that points to another node that has an edge incident on V_i.

Link j for V_i – A link that points to another node that has an edge incident on V_j.

ex:

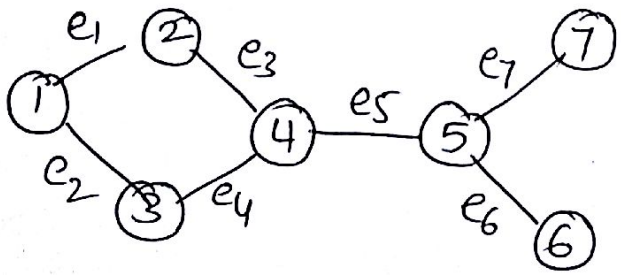
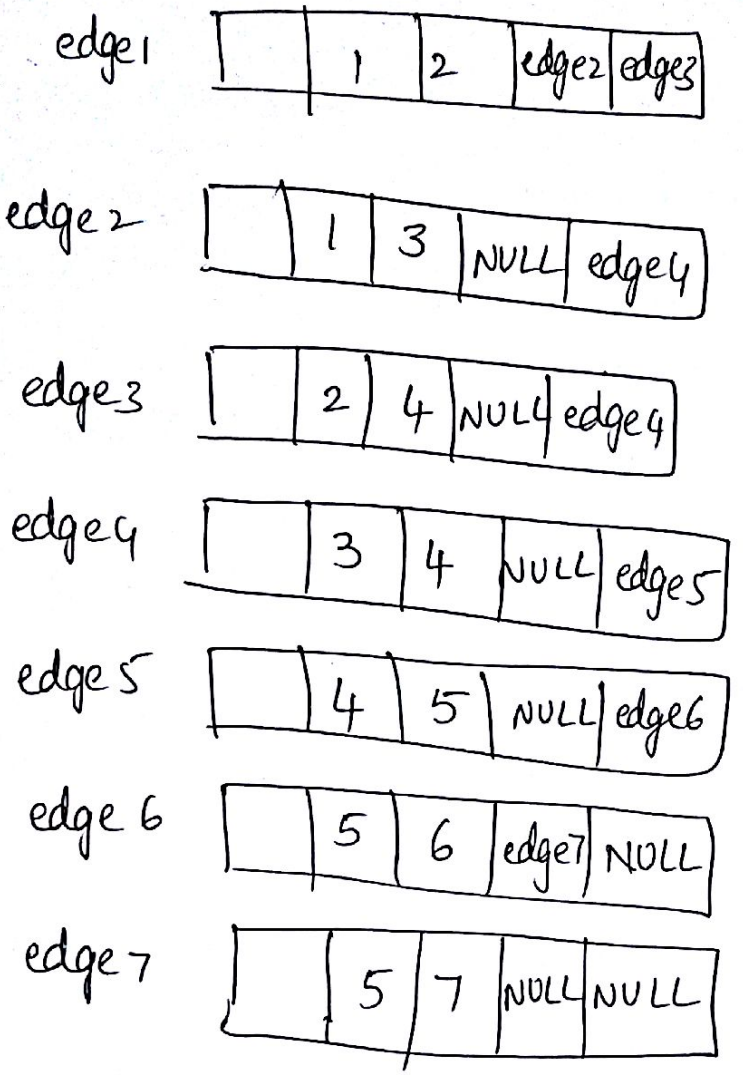


Fig: Undirected graph

The adjacency multi list for the above graph is,

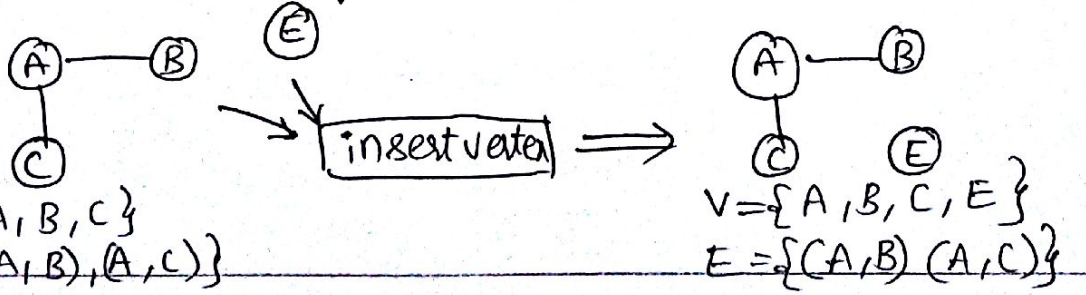


Graph Operations:

1, Insert a vertex
2, Delete a vertex
3, Add an edge
4, delete an edge
5, find a vertex
6, Traverse a graph

① Insert a vertex :- Insert vertex adds a new vertex to a graph, which is disjoint. Then it must be connected.

ex:

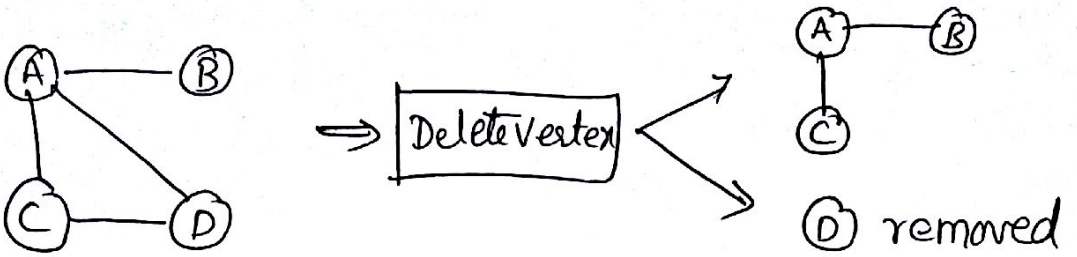


$V = \{A, B, C\}$
$E = \{(A,B), (A,C)\}$

$V = \{A, B, C, E\}$
$E = \{(A,B)\ (A,C)\}$

② Delete Vertex: It removes a vertex from the graph, & also all the connecting edges are removed.

ex:

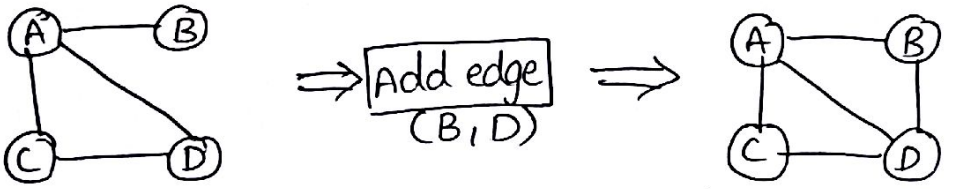


$V = \{A, B, C, D\}$
$E = \{(A,B)(A,C)(A,D),(C,D)\}$

$V = \{A, B, C\}$
$E = \{(A,B)\ (A,C)\}$

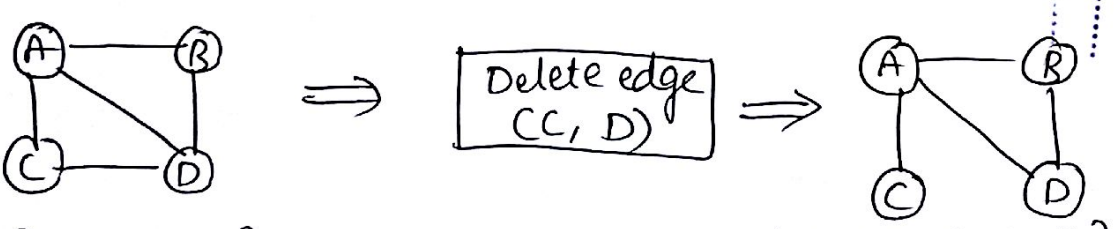
③ Add edge: It connects a vertex to a destination vertex. To add an edge, 2 vertices must be specified.

ex:



④ Delete edge: It removes one edge from a graph

ex:



$V = \{A, B, C, D\}$
$E = \{(A,B)(A,C)(A,D)(B,D)(C,D)\}$

$V = \{A, B, C, D\}$
$E = \{(A,B)(A,C)(A,D)(B,D)\}$

⑤ Find vertex: Traverse a graph & identifies the specified vertex. If the vertex is found, its data is returned. If not found, an error is indicated.

⑥ Traverse a graph: Traversing means the method of examining the nodes and edges of the graph. There are two graph traversals.

1. Breadth first Traversal (BFT)
2. Depth First Traversal (DFT)

1. Breadth First Traversal: The traversal starts from a vertex u which is said to be visited. Now all nodes v_i, adjacent to u are visited. The unvisited vertices w_{ij} adjacent to each of v_i are visited next and so on. The traversal terminates when there are no more nodes to visit. The process calls for the maintenance of a queue to keep track of the order of nodes whose adjacent nodes are to be visited.

Breadth first traversal as its name indicates traverses the successors of the start node, generation after generation in a horizontal or linear fashion. This "breadth wise" traversal is clearly visible when the traversal is worked over a graph represented as an adjacency list.

Algorithm:


```
Procedure BFT(S)
begin
    Initialize queue Q
    visited(S)=1
    call ENQUEUE(Q, S);
```

```
while ~~not~~ Queue not empty do
    call Dequeue(Q, s)
    print(s);
    for all vertices v adjacent to s do
        if (visited (v) = 0) then
        {   call ENQUEUE(Q, v).
            visited (v) = 1
        }
    end for
end while
end
```


Eg:

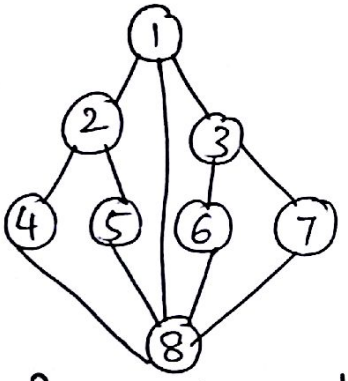


Fig: Undirected graph

Sol: Consider starting vertex as node 1

→ ENQUEUE(Q, 1) ⇒ |1|

visited [1] = 1, delete '1' from Queue

Adjacent nodes of 1 = {2, 3, 8}

→ Insert unvisited nodes into Queue

⇒ |2|3|8|

visited [2] = 1, delete '2' from Queue

Adjacent nodes of 2 = {~~1~~, 4, 5}

→ Insert unvisited nodes into Queue

⇒ [3|8|4|5]

visited [3] = 1, delete '3' from Queue

Adjacent nodes of 3 = {1, 6, 7}

→ Insert unvisited nodes into Queue

⇒ [8|4|5|6|7]

visited [8] = 1, delete '8' from Queue

Adjacent nodes of 8 = {1, 4, 5, 6, 7}

already in the queue

→ No insertion will take place

⇒ [4|5|6|7]

visited [4] = 1, delete '4' from Queue

Adjacent nodes of 4 = {2, 8}

→ No insertion will take place

⇒ [5|6|7]

visited [5] = 1, delete '5' from queue

Adjacent nodes of 5 = {2, 8}

→ No insertion will take place

⇒ [6|7]

visited [6] = 1, delete '6' from queue

Adjacent nodes of 6 = {3, 8}

→ No insertion will take place

⇒ [7]

visited [7] = 1, delete '7' from queue